A Smart Tuning Strategy for Restart Frequency of GMRES(*m*) with Hierarchical Cache Sizes

Takahiro Katagiri[1], Pierre-Yves Aquilanti[2,3], and Serge Petiton[4]

[1] Information Technology Center, The University of Tokyo,
2-11-16 Yayoi, Bunkyo-ku, Tokyo 113-8658, Japan
katagiri@cc.u-tokyo.ac.jp
[2] LIFL, Université Lille 1 Science et Technologie, Cite Scientifique, Bâtiment M3,
59655 Villeneuve d'Ascq Cedex, France
[3] A*STAR Computational Resource Centre,
1 Fusionopolis Way, #17-01 Connexis, Singapore 138632
aquilantip@acrc.a-star.edu.sg
[4] LIFL, Université Lille 1 Science et Technologie, Cite Scientifique, Bâtiment M3,
59655 Villeneuve d'Ascq Cedex, France
Serge.Petiton@lifl.fr

Abstract. In this paper, we propose a smart tuning strategy that uses the cache size hierarchy of current multicore architectures. Both increase and decrease auto-tuning (AT) strategies for the restart frequency of GMRES(*m*) (Generalized Minimum Residual) are evaluated with the proposed hierarchical cache sizes. This evaluation, using one node of the T2K Open Supercomputer (Univ. Tokyo), demonstrates that the proposed strategies are very efficient compared to previous strategies without hierarchical cache sizes. We test both strategies with 22 matrices from the University of Florida Sparse Matrix Collection. As a result, we find an average speedup of 1.13× (maximum 2.06×) using an increase strategy (an implementation of Xabclib), and an average speedup of 4.25× (maximum 15.1×) with a decrease strategy (Aquilanti's) using the proposed method.

Keywords: Auto-tuning, GMRES(*m*), Dynamic Restart Frequency Adjustment, Xabclib.

1 Introduction

Current computer architectures have complex structures, with multicore systems commonly utilizing non-uniform memory access and hierarchical caches. In terms of cache organization, several multiple caches are independent of cores, but one cache is shared across multiple cores. Thus, tuning the performance of software is becoming increasingly difficult. To solve this problem, auto-tuning (AT) technology is frequently used by non-experts to establish high performance computing on current architectures.

A wide range of problems is expressed through a linear system. Hence, solving sparse linear systems, such as the following, is a crucial task for scientific computing:

M. Daydé, O. Marques, and K. Nakajima (Eds.): VECPAR 2012, LNCS 7851, pp. 314–328, 2013.

$$Ax = b. \tag{1}$$

When the operator *A* is sparse, it is common to use iterative solvers. The Generalized Minimum Residual (GMRES) algorithm [1] is considered to be powerful and can be applied to a wide range of cases. For the iterative approximation of the solution vector, the Krylov subspace is used to determine the direction in which the solution of the linear system lies, such that:

$$x_1 \in x_0 + \mathrm{K}^m(A, r_0), \tag{2}$$

where $\mathrm{K}^m(A, r_0) \equiv span\{r_0, Ar_0, \cdots, A^{m-1}r_0\}$ is the Krylov subspace of dimension *m*, x_0 is the initial guess, x_1 is the estimated vector in the first iteration, and r_0 is the initial residual.

As GMRES iterates, its computing power and memory requirements are likely to increase when the dimension of the Krylov subspace is large. As memory is limited in practice, it is common to restart GMRES after *m* iterations. This variant is known as the restarted GMRES [1]. The parameter *m* controls the restart; hence, we call this parameter the "restart frequency." It has been demonstrated that *m* is a critical argument, driving not only memory consumption but also the execution time required for the solver to converge. Determining *m* is thus a very important issue, affecting not only high-performance libraries but also research topics in AT.

1.1 Categories of AT for the Restart Frequency of GMRES(*m*)

As the restart frequency of GMRES(*m*) is very crucial for performance, several AT strategies have been proposed. In this section, we categorize the strategies as follows.

- **Increase Strategy**

  The increase strategy is defined as follows. In the first phase, the restart frequency is assigned a small number, say *m* = 2. In the next phase, the frequency is increased using run-time information. Previous strategies in this category include that of Sosonkina *et al.* [2] and the strategy implemented in Xabclib [3]. Obviously, this strategy is good for easy problems that require only a small number of restarts to converge.

- **Decrease Strategy**

  The decrease strategy is the opposite of the increase strategy. In the first phase, the frequency is assigned a maximum size. In the next phase, the frequency is decreased using run-time information. The major strategy in this category is that of Baker *et al.* [4]. Obviously, this strategy is good for difficult problems, which require a large number of restarts. One of the drawbacks of this strategy concerns the difficulty of finding the optimal maximum size for *m*.

- **Hybrid Strategy**

  This strategy is a hybrid of the increase and decrease strategies. The frequency is dynamically increased or decreased according to run-time information. As the

hybrid strategy needs an initial restart frequency, we can define two subcategories according to whether the initial frequency increases or decreases.

The strategy proposed by Habu *et al.* [5] starts from a small initial frequency; hence, this is categorized as an increasing hybrid. On the other hand, the strategy proposed by Aquilanti *et al.* [6] starts from a maximum size, hence this is a decreasing type. As for the previous strategies, it depends on the convergence as to which type is better suited to the problem.

- **Other Considerations: The Target**

Although the strategies shown above have general properties, the target of their evaluations is limited to GMRES(*m*). There are a few implementations and preliminary evaluations for the adaptation of these strategies to other algorithms. The strategy used in Xabclib [3] has been extended to the restarted Lanczos and explicit restarted Arnoldi problems. The codes have been released to the public as a free (GNU licensed) library.

1.2 Originality of This Paper

With respect to the above categorization, we summarize the originality of this research as follows:

- **Showing Effectiveness of AT with Hierarchical Caches on Multicore Architectures**

  In general, the parameter search space in restart parameter AT is huge; hence, we need some heuristics to avoid using a brute force search and to obtain reasonable parameter settings. One of the candidates for finding reasonable parameter settings for AT is to use hardware parameters. We use an AT strategy with cache information to demonstrate the effectiveness of this approach. The key to our strategy is that the cache information helps to restrain the search of *m*.

  Aquilanti *et al.* [6] first proposed the hybrid method of decreasing type to utilize hierarchical caches for AT of the GMRES(*m*) algorithm. We re-evaluate this strategy from the following two viewpoints: (1) Supposing "real" multicore architectures with three kinds of cache, i.e., two levels of independent cache and one level of shared cache; (2) Evaluating the AT effect using an increase strategy.

- **Evaluating AT Strategies with Different Methods**

  AT methods vary amongst the different strategies. The principal method takes the current and past (one) residual vectors, and calculates the angle between them to find a stagnation point [4,5,6]. Xabclib, however, uses a method based on multiple norms from past residuals, which are taken as "sampling" points. It then uses the ratio among these sampling points, called the MM (Maximum Minimal) ratio, to find stagnation. Our research includes a comparison between the angle calculation method and the MM ratio method.

1.3 Organization of the Paper

This paper is organized as follows. In Section 2, we explain the GMRES(*m*) method and the AT strategy for the restart frequency. Section 3 describes the proposed AT strategies (increase and decrease) for hierarchical caches. We use the strategies of Aquilanti *et al.* [6] and Xabclib [3] as examples of the two categories. In Section 4, we evaluate the proposed strategies on a multicore architecture. We use one node of the T2K Open supercomputer (Univ. Tokyo), which uses the AMD Quad Core Opteron. Finally, we summarize the findings of this paper in Section 5.

2 GMRES(*m*) and AT Strategy for the Restart Frequency

2.1 The GMRES(*m*) Algorithm

The GMRES(*m*) algorithm for this paper is based on [1]. The algorithm is shown in Fig. 1.

<1> Compute $r_0 = b - Ax_0$; $\beta = \|r_0\|_2$; $v_1 = r_0 / \beta$;
<2> if ($\|r_0\|_2$ .le. ε) then goto <16>.
<3> Let the $(m+1)\times m$ matrix be $\overline{H}_m = \{h_{ij}\}_{1\le i\le m+1, 1\le j\le m}$. Set $\overline{H}_m = 0$.
<4> *do* $j = 1, m$
<5> Compute $\omega_j = Av_j$
<6> *do* i = 1, *j*
<7> $h_{ij} = (\omega_j, v_j)$
<8> $\omega_j = \omega_j - h_{ij} v_j$
<9> *enddo*
<10> $h_{i+1,j} = \|\omega_j\|_2$. If $h_{i+1,j} = 0$ then Set $m = j$; goto <12>
<11> $v_{j+1} = \omega_j / h_{j+1,j}$
<12> *enddo*
<13> Let the $\{v_1, v_2, ..., v_m\}$ be V_m.
<14> Compute y_m to minimize $\|\beta e_1 - \overline{H}_m y\|_2$; $x_m = x_0 + V_m y_m$;
<15> $x_0 = x_m$; goto <1>;
<16> continue

Fig. 1. GMRES(m) algorithm

2.2 AT Strategies for the Restart Frequency of GMRES(*m*)

To perform AT on the restart parameter *m* in Fig. 1, we append the following to line <1>:

<1-1> r_{prev} is set to previous residual vector. If this is the first iteration, then set $r_{prev} = r_0$.
<1-2> Line <1> in Fig.1.
<1-3> $r_{cul} = r_0$; $m = f_{AT}(r_{cul}, r_{prev})$;

The function $f_{AT}(r_{cul}, r_{prev})$ forms the AT strategy.

We first take the Xabclib strategy, which is categorized as an increase strategy. The definition of $f_{AT}(r_{cul}, r_{prev})$ is shown in Fig. 2.

```
<0> SAMP = |r_cul|_2
<1>CALL OpenATI_DAFRT
       (5, SAMP, IRT, IATPARAM, RATPARAM, INFO)
<2> if (IRT .EQ. 1) then
<3>    MOLD = M
<4>    M = M + IATPARAM(5)
<5>    if (M .GT. MSIZE) then
<6>       M = MSIZE
<7>    endif
<8> endif
<9> return M
```

Fig. 2. Restart Frequency AT for Xabclib (Increase Strategy). In this example, the **OpenATI_DAFRT** finds stagnation using the norms of the past five residual vectors. The "5" is an AT parameter. If stagnation is found, the frequency is increased by IATPARAM(5), which is set to 5 as a default value. The increase value "5" is also a tunable parameter.

Note that the stagnation state is found via the API of **OpenATI_DAFRT**, which is provided by OpenATLib [3]. In this example, it requires the past five residual norms. The number of past residual norms is a tunable parameter. The default implementation of Xabclib is to use five points.

To demonstrate a decrease strategy, we consider Aquilanti's method (Fig. 3). There are also tunable parameters in this strategy: maximum frequency (TRestart%m_max), default frequency (TRestart%m_def), and maximum count of decrease cycles (TRestart%m_count_max).

```
<0> resid = |r_cul|_2 ; presid = |r_prev|_2 ;
<1> max_cr = cos(8.*PI/180.); min_cr = cos(80.*PI/180.);
<2> cr = resid / presid      !! get the angle
<3> if (cr .gt. max_cr) then   !! normal cycling
<4>    M = TRestart%m_max
<5> else
<6>    if ( (cr .lt. min_cr) .or. (TRestart%m_count_max
           .lt. TRestart%m_count) ) then    !! enter an aug cycle
<7>      if (TRestart%m_aug .eq. 0) then
<8>         TRestart%m_aug = 1; TRestart%m_floor = 1;
<9>      else    !! or continue it
<10>        TRestart%m_floor = TRestart%m_floor + 1
<11>     endif
<12>     TRestart%m_count = 0;
<13>     M = TRestart%m_floor * TRestart%m_max
<14>  else
<15>     if (M - TRestart%m_incr .ge. TRestart%m_min) then
<16>        M = M - TRestart%m_incr
<17>     else
<18>        M = TRestart%m_max
<19>     endif
<20>     if ((TRestart%m_aug .eq. 1) .and. (TRestart%m_count .le.
<21>        TRestart%m_count_max) ) then   !! if in aug cycle
<22>        if (TRestart%m_def * TRestart%m_floor
                .lt. TRestart%m_max) then
<23>        M = TRestart%m_def * TRestart%m_floor
<24>        TRestart%m_count = TRestart%m_count + 1
<25>     else
<26>        M = TRestart%m_def;   TRestart%m_aug = 0;
<27>     endif
<28>  endif
<29> endif
<30> return M
```

Fig. 3. Restart Frequency AT for Aquilanti's (Decrease) Strategy. The maximum frequency (TRestart%m_max), default frequency (TRestart%m_def), and maximum count of decrease cycles (TRestart%m_count_max) are tunable parameters in this strategy.

3 A Smart Tuning Strategy with Hierarchical Cache Sizes

3.1 Using the Vector Size of Caches to Better Estimate *m*

Most of the computational cost of GMRES(*m*) is due to the orthogonalization process in lines <4>–<12> in Fig. 1. The computational complexity of orthogonalization depends on the restart frequency *m*, i.e., $O(nm^2)$, where *n* is the dimension of matrix *A*.

Orthogonalization is performed in a space of dimension $n \times m$. The orthogonalization process can be parallelized by threads based on the rows of the space. (See parallel Classical Gram–Schmidt or Modified Gram–Schmidt procedures.) With respect to parallel implementation with threads, we can estimate a better value m^* with a double-precision computation using the following formula:

$$m^* = Memory\ Size / 8n / The\ number\ of\ threads, \tag{3}$$

where the Memory Size corresponds to the sizes of the L1 cache (Independent), L2 cache (Independent), and L3 cache (Shared), etc. The memory size of the sparse matrix *A* is not considered in this model. It will, however, give a good estimate for the orthogonalization complexity, as all computations are performed with vectors that must be orthogonalized. In addition, if *m* is large, the most demanding process of GMRES(*m*) will be the orthogonalization. With this in mind, we consider this to be a reasonable estimation for *m**.

3.2 Principle of AT Using Hierarchical Caches

We use one socket of the AMD Opteron (Barcelona) to explain AT with hierarchical caches. The AMD Opteron has three types of cache: L1 cache (Independent, 64 KB), L2 cache (Independent, 512 KB), and L3 cache (Shared between four cores, 2 MB). Taking into account the real configuration of the caches, the idea to improve the AT strategy is summarized as follows:

- Use cache information to set maximum values of *m* for the AT. In the AMD Opteron case, the following hierarchy is formed (although this is not necessarily limited to the specific configuration):
  - M_MAX_{L1} : maximum size of *m* on the L1 cache.
  - M_MAX_{L2} : maximum size of *m* on the L2 cache.
  - M_MAX_{L3} : maximum size of *m* on the L3 cache.
  - M_MAX_{MM} : maximum size of *m* on the main memory.

3.3 AT for an Increase Strategy with Cache Hierarchy

Fig. 4 shows the proposed AT method with cache hierarchy for the Xabclib strategy.

```
<3-1>   if (MLEVEL .eq. 1) then
<3-2>     M = M_MAX_L1; MLEVEL = 2;
<3-3>     else if (MLEVEL .eq. 2) then
<3-4>       M = M_MAX_L2; MLEVEL = 3;
<3-5>     else if (MLEVEL .eq. 3) then
<3-6>       M = M_MAX_L3; MLEVEL = 4;
<3-7>    else
<3-8>       M = M + IATPARAM(5)
<3-9>    endif
<3-10> if (M .GT. MSIZE) then
<3-11>     M = M_MAX_MM
<3-12> endif
```

Fig. 4. Increase strategy (Xabclib) with cache hierarchy. Additions to line <3> in Fig. 2 are shown. Set MLEVEL = 1 before the main loop of GMRES(*m*) in Fig. 1.

3.4 AT for a Decrease Strategy with Cache Hierarchy

Fig. 5 shows the proposed AT method with cache hierarchy for Aquilanti's strategy.

```
<3'> m = TRestart%m_maxs(mlevel)
<4'> if (TRestart%mlevel .le. 3) TRestart%mlevel = TRestart%mlevel + 1
     else TRestart%mlevel = 1
<13'> m = TRestart%m_floor * TRestart%m_maxs(TRestart%mlevel)
<18'> m = TRestart%m_maxs(TRestart%mlevel)
<22'> if (TRestart%m_maxs(TRestart%mlevel) * TRestart%m_floor .lt.
        TRestart%m_max) then
<23'>   m = TRestart%m_maxs(TRestart%mlevel) * TRestart%m_floor
<26'> m = TRestart%m_maxes(TRestart%mlevel); TRestart%m_aug = 0;
```

Fig. 5. Decrease strategy (Aquilanti's) with cache hierarchy. The modifications to lines in Fig. 3 are shown. Set TRestart%mlevel = 1 before the main loop of GMRES(*m*) in Fig. 1.

3.5 Implementation Variants of the Decrease Strategy

There are some variants to the cache hierarchy strategy for Aquilanti's method. Roughly speaking, there are three ways to adapt the cache sizes: (1) using the maximum value; (2) using the minimum (default) value; or (3) a mixture of both.

We take option (3)—the maximum value for the cache hierarchy in the current level (see lines <3'> and <18'> in Fig. 5) and the default values for current level (see line <26'> in Fig. 5). The reason for this approach is to prevent setting too small a default size with respect to the execution on the previous level.

At the end, we add the following parameters to the original strategy:

- m_max = M_MAX_{MM}
- TRestart%m_maxs(1) = M_MAX_{L1}
- TRestart%m_maxs(2) = M_MAX_{L2}
- TRestart%m_maxs(3) = M_MAX_{L3}
- TRestart%m_maxs(4) = 200 (this is m_max, which is the maximum frequency parameter in Aquilanti's original strategy)

4 Numerical Experiments

4.1 Computer Environment

We used the T2K Open Supercomputer, which is a HITACHI HA8000 installed at the Information Technology Center, University of Tokyo. Each node contains four sockets of the AMD Opteron 8356 (Quad core, 2.3 GHz). The L1 cache is 64 KB/core, the L2 cache is 512 KB/core, and the L3 cache is 2 MB/4 cores. The memory on each node is 32 GB with DDR2-667 MHz. The theoretical peak is 147.2 GFLOPS/node. The inter-node connection comprises four lines of the Myri-10G with a full bisection connection. This attains 5 GB/s in both directions. We used the Intel Fortran Compiler Professional Version 11.0 with options "-O3 -m64 -openmp –mcmodel=medium."

4.2 Experimental Conditions

We used a pre-release version of Xabclib ver.1.0 [3] for the GMRES(m) implementation of both strategies. The GMRES(m) subroutine on Xabclib is **OpenATI_GMRES**. ILU(0) was chosen as a preconditioner. The convergence tolerance was set to 1.0e-08, and the time tolerance was set to 600 s. If an iteration had not converged after 600 s, the routine was forcibly stopped. This is the fundamental function of Xabclib.

We formed a solution vector x whose elements were set to 1. The right-hand-side (RHS) vector was then generated by Ax. The initial guess x_0 was set to 0.

For Aquilanti's original strategy (Fig. 3), we required a number of default parameters, which were set as follows in this experiment.

- m_def = 20
- m_min = 3
- m_max = 200
- m_incr = 3
- m_count_max = 5

4.3 Test Matrices

We used 22 non-symmetric, real matrices from the University of Florida Sparse Matrix Collection (referred to hereafter as the UF collection) [7]. The test matrices are shown in Table 1. Equation (3) was used to calculate the cached *m** size.

According to Table 1, the cached *m** sizes for L2 and L3 are the same. This is because the size of the shared L3 per core is the same as the L2 (512 KB) when we use 16 cores on the AMD Opteron. We set the cached sizes of *m** to M_MAX_{L1}, M_MAX_{L2}, and M_MAX_{L3}, respectively.

Table 1. Test matrices from the UF collection and cached *m** sizes on the AMD Quad Core Opteron. N is the dimension of the matrix, and NNZ is the number of non-zero elements.

Name	N	NNZ	L1 Cached *m** Size	L2 Cached *m** Size	L3 Cached *m** Size (16 cores)
chipcool0	20082	281150	6.5	52.2	52.2
chem_master1	40401	201201	3.2	26.0	26.0
torso1	116158	8516500	1.1	9.0	9.0
torso2	115967	1033473	1.1	9.0	9.0
torso3	259156	4429042	0.5	4.0	4.0
memplus	17758	126150	7.4	59.0	59.0
ex19	12005	259879	10.9	87.3	87.3
poisson3Da	13514	352762	9.7	77.6	77.6
poisson3Db	85623	2374949	1.5	12.2	12.2
airfoil_2d	14214	259688	9.2	73.8	73.8
viscoplastic2	32769	381326	2.0	32.0	32.0
xenon1	48600	1181120	1.3	21.6	21.6
xenon2	157464	3866688	0.4	6.7	6.7
wang3	26064	177168	2.5	40.2	40.2
wang4	26068	177196	2.5	40.2	40.2
ecl32	51993	380415	1.3	20.2	20.2
sme3Da	12504	874887	5.2	83.9	83.9
sme3Db	29067	2081063	2.3	36.1	36.1
sme3Dc	42930	3148656	1.5	24.4	24.4
epb1	14734	95053	4.4	71.2	71.2
epb2	25228	175027	2.6	41.6	41.6
epb3	84617	463625	0.8	12.4	12.4

4.4 Results and Discussion

Effect on the Increase Strategy (Xabclib)

Fig. 6 shows the effect of AT with hierarchical cache sizes for the increase strategy (Xabclib) on the T2K (16 threads).

According to Fig. 6, the performance for several matrices is improved. The average speedup of execution time compared to the original is 1.13×. In particular, the airfoil_2d matrix from the UF collection establishes a speedup of more than 2×. We examined the evolution of the restart frequency for this case. Fig. 7 illustrates the change in *m* using the original Xabclib method and our proposed cache size AT strategy for airfoil_2d.

According to Fig. 7, the frequency of *m* is increased dramatically, from 9 to 73, after the first restart process. This is because airfoil_2d is not a large matrix; the 73 vectors are all cached in L2. Hence, after the first restart, our strategy can suddenly increase *m* up to 73. This causes faster convergence than in the original strategy.

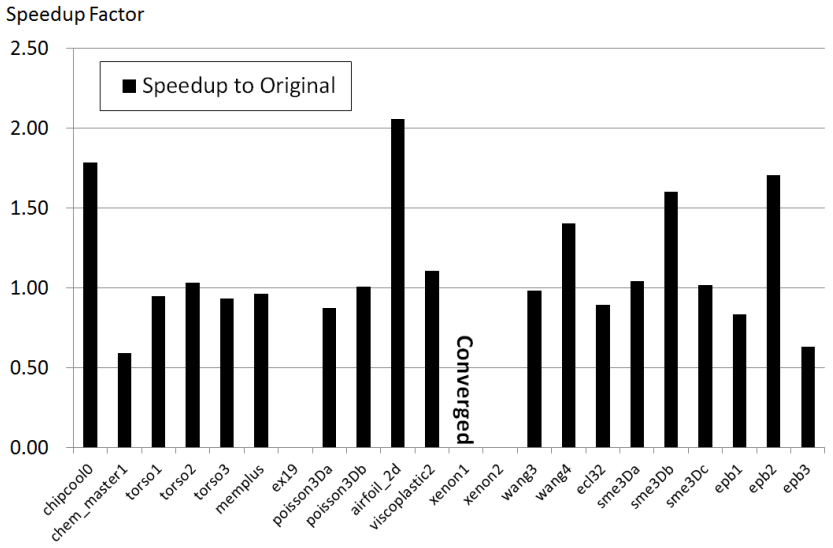


Fig. 6. Effect of AT with hierarchical cache sizes for the increase strategy (Xabclib) on the T2K (16 threads). Execution time of the original is normalized to 1. Speedup factor greater than 1 implies faster convergence than the original. The ex19 and xenon2 matrices do not converge in the ILU(0) preconditioner, whereas xenon1 converges if we use the cache size strategy.

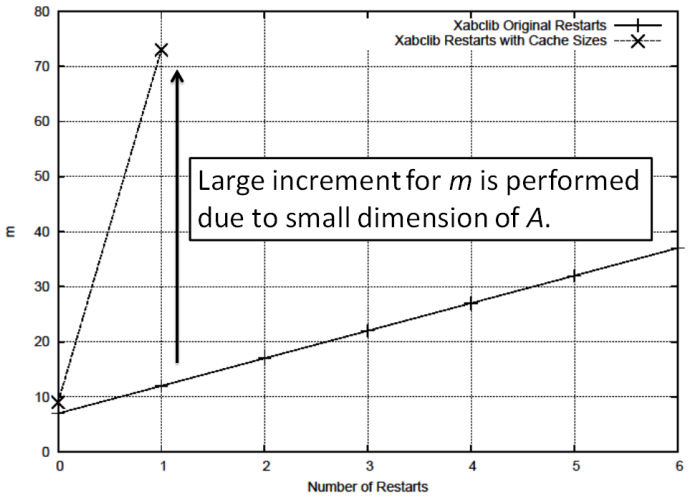


Fig. 7. Change in restart frequency using the Xabclib strategy in the airfoil_2d

Effect on the Decrease Strategy (Aquilanti's)

Fig. 8 shows the effect of AT with hierarchical cache sizes for the decrease strategy (Aquilanti's) on the T2K (16 threads).

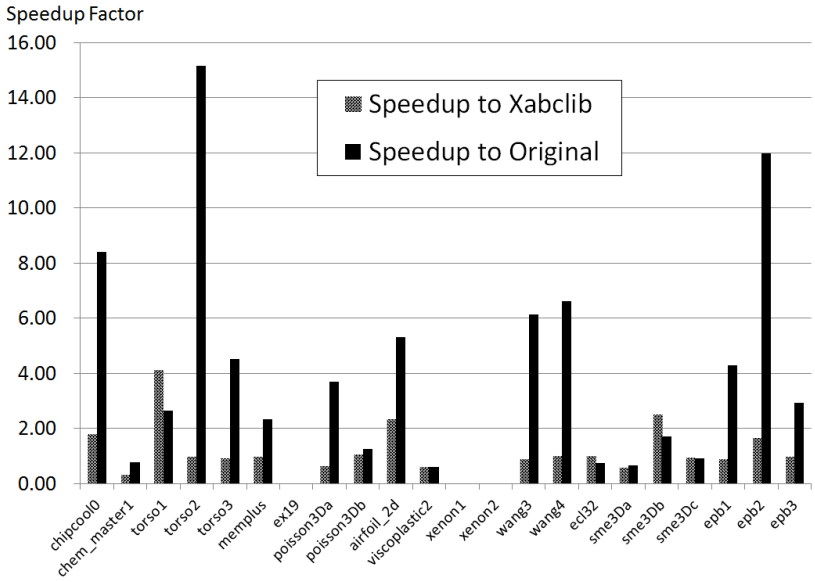


Fig. 8. Effect of AT with hierarchical cache sizes for the decrease strategy (Aquilanti's) on the T2K (16 threads). Execution time of the original (Aquilanti's) is normalized to 1. Speedup factors greater than 1 implies faster convergence than the original. Speedup compared to the original Xabclib increase strategy is also shown. The ex19, xenon1, and xenon2 matrices did not converge in the ILU(0) preconditioner.

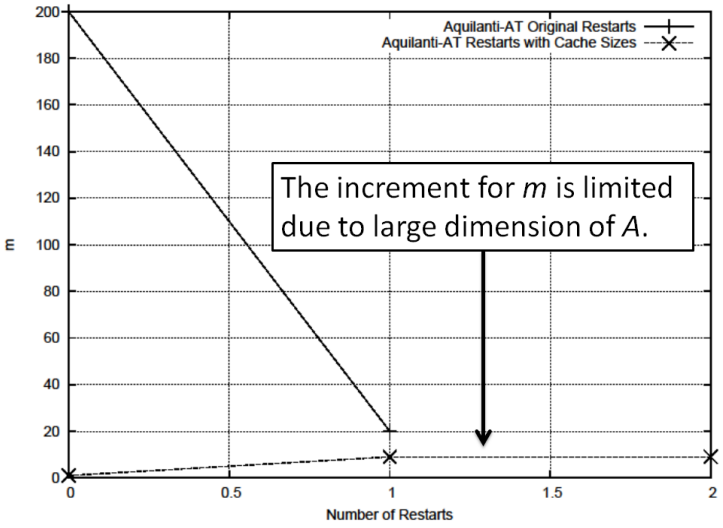
According to Fig. 8, the performance for several matrices is strongly improved. The average speedup factor is 4.25×. In addition, the average speedup compared to the original Xabclib strategy is 1.29×, which is considerable. Therefore, this implies that the crucial effect is due to our cache size strategy.

The torso2 matrix, in particular, established a speedup of more than 15× the original decrease strategy, and torso1 achieved a 4× speedup compared to the original Xabclib increase strategy. We examined the evolution of the restart frequency for both of these matrices, and have plotted these in Fig. 9 in order to help explain these phenomena.

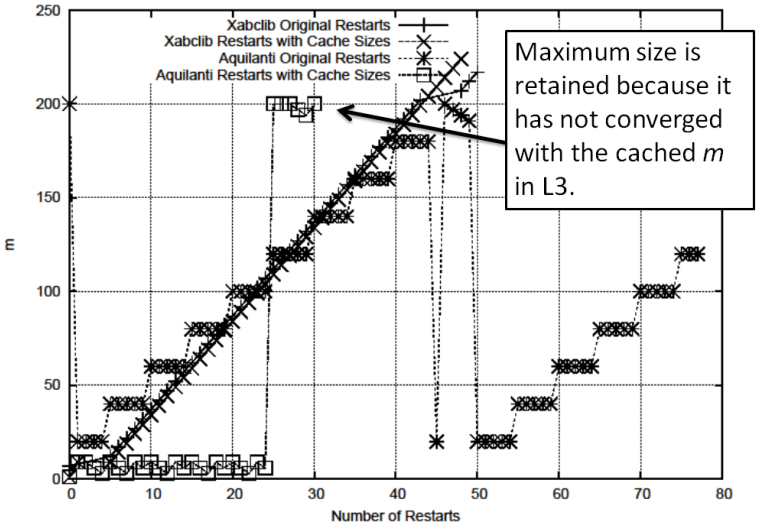
According to Fig. 9 (a), the maximum frequency in the proposed strategy is limited, as it still uses the cached m sizes for L1 and L2. The torso2 matrix is large, hence the cached m for L2 is only 9, whereas the original strategy uses m = 200, the default maximum. In addition to this, torso2 needs a very small value of m to converge. This fact leads to the enormous speedup compared to the original.

In contrast, torso1 is a difficult problem in that it requires almost maximum frequency, i.e., 200, to converge. In the first phase, the cache size strategy is trying to

find better values for *m* within L1, L2, and L3; however, this does not give convergence because *m* is very small i.e., 9, in this case. After searching all cache sizes, the strategy retains the default maximum size, i.e., 200. On the other hand, the Xabclib strategy is to increase *m* step-by-step until it reaches 200. This causes very slow convergence compared to Aquilanti's strategy.



(a) torso2



(b) torso1

Fig. 9. Change in restart frequencies for Aquilanti's strategy on (a) torso2 and (b) torso1

From the above discussions concerning the statistically significant gains in speedup, we can conclude that our proposed cache size strategy is crucial to AT for both increase and decrease strategies.

It is difficult to implement a good decrease strategy without cache information; this is because it is generally difficult to set a better maximum restart frequency. Using the cache size strategy, it is clear that no additional cost is needed to set the maximum value. As a result, the decrease strategy is faster than the increase strategy in terms of average speedup.

5 Conclusion

In this paper, we proposed a smart tuning strategy using hierarchical cache sizes for a current multicore architecture. We have proposed an auto-tuning (AT) strategy for the restart frequency of both increase and decrease GMRES(*m*) methods.

As a result of performance tuning using one node of the T2K Open Supercomputer composed of an AMD Quad Core Opteron (16 cores), the proposed AT strategies were found to be very efficient compared to the original strategies without hierarchical cache sizes.

We evaluated the proposed strategies on 22 matrices from the University of Florida Sparse Matrix Collection. The results showed an average speedup of 1.13× for the increase method (an implementation of Xabclib) and an average speedup of 4.25× for the decrease method (Aquilanti's) using the proposed strategy.

One of the drawbacks to the traditional decrease strategy is the difficulty in determining the optimal maximum restart frequency—if too large a value is specified, the algorithm takes a long time; however, if too small a value is specified, it may not converge at all. According to the results of our numerical experiment, we found that the performance of the decrease strategy was improved by a factor of 15. This was caused by the selection of appropriate values for the maximum restart frequency based on cache size information.

In addition, we found that the decrease strategy is better than the increase strategy, in terms of the average speedup, if the maximum frequency is set appropriately. The hierarchical cache information is a crucial factor in setting an appropriate maximum frequency.

As the L2 and L3 cache sizes are the same when we use 16 cores for the AMD Quad Core Opteron, the evaluation of several multicore architectures is important future work. The proposed strategy does not permit "multiple" re-use of the cache information. Constructing such a strategy is also vital in future research.

Acknowledgments. This work is supported by the FP3C "Framework and Programming for Post Petascale Computing" Project, funded by Strategic International Research Cooperative Program, JST, Japan, and Agence National de la Recherche (ANR), France. The authors would like to thank members of the Xabclib project for the use of a pre-released version of Xabclib ver.1.0.

References

1. Saad, Y., Schults, M.: GMRES: A Generalized Minimal Residual Algorithm For Solving Nonsymmetric Linear Systems. SIAM J. Sci. Stat. Comput. 7(3), 856–869 (1986)
2. Sosonkina, M., Watson, L., Kapania, R.: A New Adaptive GMRES Algorithm for Achieving High Accuracy. Technical Report, Computer Science, Virginia Polytechnic Institute and State University (1996)
3. Xabclib Project, http://www.abc-lib.org/Xabclib/index.html
4. Baker, A., Jessup, E., Kolev, Tz.: A Simple Strategy for Varying The Restart Parameter in GMRES(m). Journal of Computational and Applied Mathematics 230(2), 751–761 (2009)
5. Habu, M., Nodera, T.: GMRES(m) Method with Changing The Restart Cycle Dynamically. Trans. IPS Japan 43(6), 1795–1803 (2002) (in Japanese)
6. Aquilanti, P.-Y., Petiton, S., Calandra, H.: Parallel Auto-tuned GMRES Method to Solve Complex Non-Hermitian Linear System. In: Proceedings of iWAPT 2010 (2010)
7. The University of Florida Sparse Matrix Collection, http://www.cise.ufl.edu/research/sparse/matrices/